02

Overview: Checked y unchecked

Transcripción

[00:00] Sean bienvenidos a la quinta clase de su curso de Java: el módulo de Excepciones. Ya hasta este momento ya hemos visto cómo es que Java lanza los diversos errores que podemos tener en el código, sabemos pues que Java agrupa estas excepciones en dos grandes grupos que son las excepciones checked y unchecked, que son digamos pues las verificadas y no verificadas.

[00:32] ¿Y qué significa esto de verificar y no verificar? Haciendo un overview de lo que hemos visto la clase pasada, una excepción verificada, una del tipo check, es una excepción que, valga la redundancia, es verificada por el compilador para que nosotros tratemos esa excepción a nivel del código de forma obligatoria.

[00:56] Por ejemplo, si mi excepción verificada es declarada en la firma de un método, entonces ahí yo sí o sí, de cualquier forma necesito que los métodos que van a llamar al método que está lanzando una excepción sepan cómo tratar con ese tipo de error, caso contrario de una excepción no verificada, que es la unchecked.

[01:19] Estas excepciones extienden de RuntimeException, y pueden ser excepciones que pueden ocurrir o puede que no, pero si es que ocurren, esperemos estar preparado para para poder tratar este tipo de error. Entonces las excepciones no verificadas extienden de RuntimeException, mientras que las verificadas extienden de Exception.

[01:47] Es por esta razón, pues que no podemos hacer una extensión directamente de Exception, si no obligaríamos pues a al programador a sí o sí hacer tratamiento de cada excepción que vayamos lanzando. Y no es también el objetivo de Java. Por otro lado, solamente mencionando tenemos el grupo de los errores. ¿Qué son los errores?

[02:06] Los errores son, valga la redundancia, los errores lanzados por la máquina virtual de Java para situaciones en las cuales el programador no puede tener ningún tipo de control, por ejemplo falta de memoria, que la máquina virtual de Java crasheó o alguna cosa así. Entonces, esos son los errores.

[02:31] Como programadores no tenemos acceso a tratar errores ni a lanzar errores, eso es hecho por la misma máquina virtual de Java, la JVM. Pero sí tenemos control sobre las excepciones, podemos crear nuestras propias excepciones, como lo hemos hecho, y definirlas como excepciones verificadas o no verificadas, según pues necesitemos.

[02:50] Ahora, con este overview, vamos a volver aquí al código. Y si recuerdan, nosotros creamos una excepción nuestra, que es mi excepción y está extiende de RuntimeException. ¿Qué significa? Que es una excepción no verificada. Yo voy a cambiar esto aquí, para una excepción verificada, entonces yo quiero obligar aquí a que mi Exception sea tratada en el código.

[03:19] Y para esto, solo para recrear lo que hemos venido haciendo, a modo de repaso, yo voy a crear aquí un void deposita en la clase cuenta que hemos creado, recuerden que esta clase cuenta no tiene nada, y yo voy a crear un método aquí de cero. Tengo el método aquí en visibilidad por defecto. Por ahora no me voy a preocupar por eso.

[03:45] ¿Y cómo es que yo obligo a un método a verificar una excepción? ¿Cómo es que yo obligó al código a que haga verificación de una excepción del tipo checked? Con la palabra reservada throws, que está aquí. ¿Y qué voy a lanzar

ahí? ¿Qué excepción va a lanzar este método? MiException, perfecto. "Ctrl + espacio" y autocompletamos.

[04:10] ¿Cómo vamos a probar esto? Vamos a darle un New class y aquí le vamos a escribir TestCuentaExceptionChecked, porque vamos a testear una excepción verificada. Le damos finish. Venimos aquí y creamos nuestro ya conocidísimo método main, recuerden, "Ctrl + espacio", nos va a sugerir el método main, le damos enter y él va a autogenerar ese código por nosotros. Perfecto.

[04:42] ¿Y qué voy a hacer aquí? Voy a instanciar una variable tipo cuenta, entonces voy a decirle cuenta, cuenta, = new, "Ctrl + espacio", cuenta. Perfecto. ¿Y qué voy a hacer? Voy a llamar al método deposita. Y aquí ya viene, digamos, pues el primer error. ¿Por qué? Él aquí no me está dejando compilar porque obviamente, él me está diciendo, tú estás declarando en este método que él va a lanzar MiException.

[05:19] Y MiException es una excepción del tipo checked, es verificada, entonces el compilador necesita de cualquier forma, asegurarse que nosotros estamos tratando este error, cosa diferente si yo tuviera esto. RuntimeExcepcion, guardo aquí y vemos que él compila tranquilamente. ¿Por qué?

[05:41] Por más que yo estoy lanzando aquí esta excepción o estoy diciendo que este método va a lanzar esta excepción, como es unchecked, no verificada, entonces el compilador no se va a importar, si es que yo estoy haciendo un tratamiento o no de esta excepción. Para seguir con el ejemplo, yo lo voy a dejar solamente como exception.

[06:09] Voy a borrar, aquí voy a guardar nuevamente, él vuelve a dar ese error. Y para esto lo que yo voy a hacer, el id aquí de Eclipse, aquí me da dos sugerencias, la primera es o añades la declaración throws en la firma del método, en este caso, el método main, o lo envuelves con una estructura

try/catch, que es la que hemos venido usando para tratar, pues las excepciones. Yo voy a escoger la segunda.

[06:36] Yo podría hacerlo de dos formas, podría escribir aquí try, abrir el bloque, y poner este método aquí adentro deposita, recuerden que un try no puede vivir sin su catch, ¿y en el catch yo qué excepción voy a atrapar? MiException. Y su nombre variable, ex. Y aquí le puedo poner System.out.println("Exception atrapada"). Perfecto. ¿Cuál es la segunda forma?

[07:11] Primero, vemos que aquí ya está compilando bien porque yo estoy haciendo un tratamiento de la excepción. Pero si yo no quisiera escribir el código, entonces lo que yo voy a hacer es voy a sacar esto de aquí, voy a borrar este try/catch porque no lo voy a usar. Supongamos que no he hecho nada, yo puedo autogenerar ese código escogiendo la segunda opción Surround with try/Catch.

[07:34] Le doy clic, ¿y automáticamente qué hizo? Básicamente el mismo código que yo ya había escrito, él me lo ha autogenerado. Y aquí lo que él está haciendo, que es en mi caso, yo imprimí manualmente vía consola la excepción, pero él está llamando directamente al método printStackTrace, que ya lo hemos visto anteriormente, ya sabemos para qué sirve, de la clase throwable.

[07:59] Él nos va a dar información de todo el StackTrace de la excepción lanzada. Entonces, acabamos de ver que necesitamos atrapar esta excepción de cualquier forma, si es que la excepción es del tipo checked. Si fuera una excepción del tipo unchecked, entonces simplemente el compilador no se va a hacer ningún problema, me va a dejar compilar y si en algún momento yo lanzo la excepción, bueno yo debería haber estado ya preparado para tratarla.

[08:34] Pero no va a ser un requisito para compilar mi programa y vemos que aquí una clase, la clase flujo, dejó de compilar. ¿Por qué dejó de compilar? Porque método 1 está lanzando MiException, y MiException yo la he vuelto una

excepción del tipo checked. En el siguiente video vamos a solucionar este problema de aquí. Nos vemos.